MEDICAL INSURANCE COST PREDICTION USING MACHINE LEARNING

CONTENTS

1. INTRODUCTION

People's healthcare cost forecasting is a valuable tool for improving healthcare accountability. The healthcare sector produces a huge amount of data related to patients, diseases, and diagnoses. Still, since it has not been analyzed properly, it does not provide the significance that it holds along with the patient healthcare cost.

A health insurance company can only make money if it collects more than it spends on the medical care of its beneficiaries. On the other hand, even though some conditions are more prevalent for certain segments of the population, medical costs are difficult to predict since most money comes from rare conditions of the patients. The objective of this article is to accurately predict insurance costs based on people's data, including age, Body Mass Index, smoking or not, etc. Additionally, we will also determine what the most important variables influencing insurance costs are. These estimates could be used to create actuarial tables that set the price of yearly premiums higher or lower according to the expected treatment costs. This is a regression problem.

The first step is to analyze the dataset. Then we perform data preprocessing. The dataset is divided into training and testing data. We use various algorithms for training the model. The training data teaches what the expected output looks like and the testing data is used to evaluate the regression model.

The algorithm used in this model is Random Forest Regressor, Linear Regression, and Decision Tree Regression.

2. CHAPTER 2
2.1 DATA SET DESCRIPTION

- The original dataset is available on Kaggle website. In the dataset, there are 7 columns and 1338 rows. Out of the 7 columns, 3 are categorical data and 4 are numerical data.
- The columns are:-

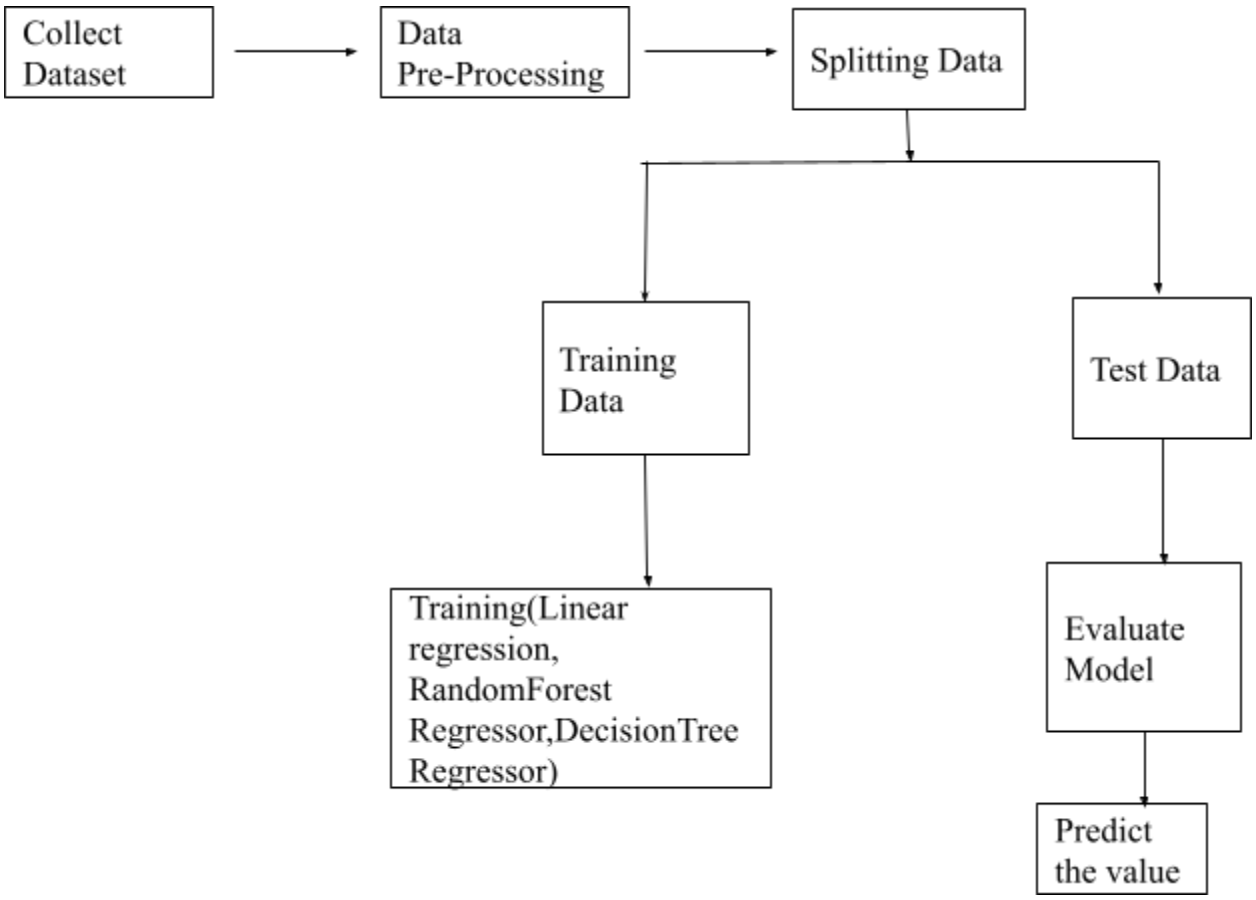
1. age: Age of client. It has an integer value. It contains values from the range 18 to 64.
2. sex: Gender of the client: Female or Male.
3. BMI: It denotes Body Mass Index.BMI indicates weights that are exceptionally high or low in relation to height.
4. children: Number of children the client has. It has an integer value. It has values from the range 0 to 5.
5. smoker: Smoking state: Smoking or not.
6. region: The beneficiary's residential area in the US: northeast, southeast, southwest, and northwest.
7. charges: Individual medical costs billed by health insurance. It has an integer value. It has values from the range 1211.873 to 63770.428

- Since we are predicting insurance costs, charges will be our target feature.
- The independent variables are ages, sex,bmi, children, smoker, and region and the dependent variable is the charge.
- We convert the data of the categorical columns into numerical data.

Sex	0:male 1:female
Smoker	0:yes 1:no
Region	0:southeast 1:southwest 2:northeast 3:northwest

2.2 METHODOLOGY

We have performed machine learning techniques on medical insurance data. The medical insurance cost dataset is gained from KAGGLE's website, and we performed the data preprocessing. After preprocessing, we select the features by performing feature engineering. Then, the dataset is split into two parts, train and test datasets; about 80% of the total data are used for training, while the rest is for testing. The training dataset is used to create a model that predicts medical insurance costs, while the test dataset is used to evaluate the regression models. For regression exploring the dataset, then categorical values are converted to numerical values.



3. CHAPTER 2
3.1 ALGORITHMS USED IN OUR PROJECT

RANDOM FOREST REGRESSOR

Random forest is a supervised machine learning algorithm used to solve classification as well as regression problems. It is a type of ensemble learning technique in which multiple decision trees are created from the training dataset and the majority of output from them is considered the final output.
Random forest is a very popular technique due to its simplicity and ability to produce robust results.

WORKING OF THE ALGORITHM :

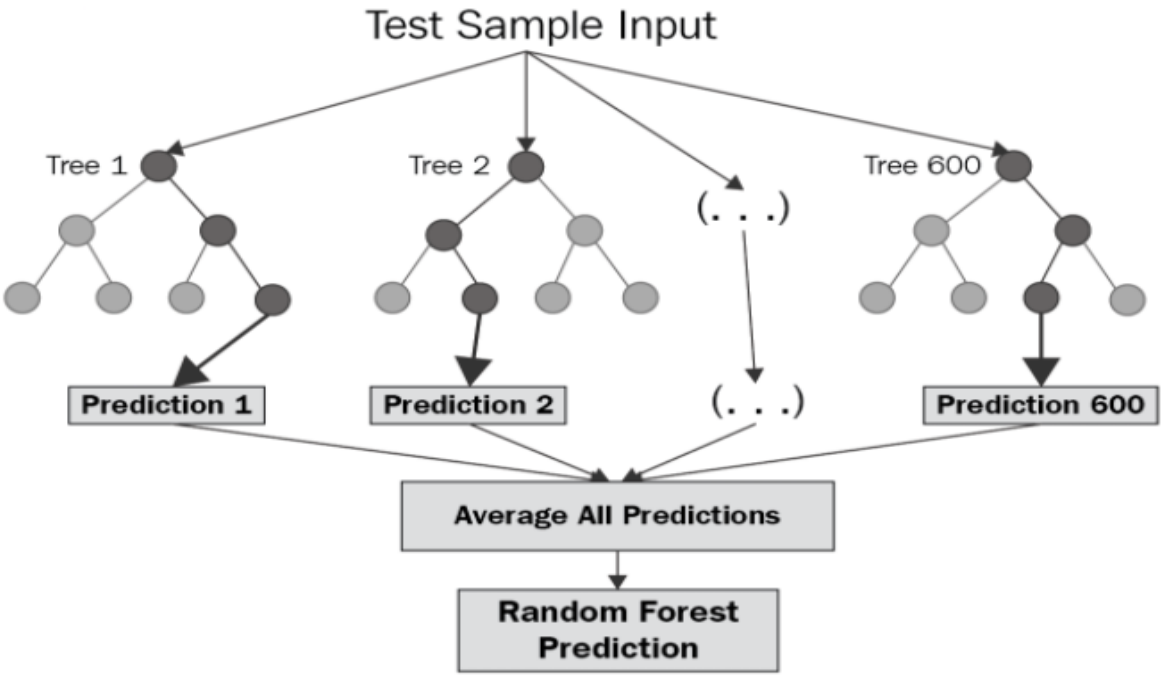
Random Forest works in two-phase first is to create the random forest by combining N decision trees, and the second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram
Step 1: In Random forest n number of random records is taken from the data set having k number of records.

Step 2: Individual decision trees are constructed for each sample.

Step 3: Each decision tree will generate an output.

Step 4: Final output is considered based on Majority Voting or Averaging for Classification and regression respectively



ADVANTAGES:-

- It is one of the most accurate learning algorithms available. For many data sets, it produces a highly accurate classifier.
- It runs efficiently on large databases.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.
- It generates an internal unbiased estimate of the generalization error as the forest building progresses.
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.

DISADVANTAGES:-

- It requires much computational power as well as resources as it builds numerous trees to combine their outputs.
- It also requires much time for training as it combines a lot of decision trees to determine the class.
- Due to the ensemble of decision trees, it also suffers interpretability and fails to determine the significance of each variable.

We implemented the random forest algorithm because random forest regression is great with high-dimensional data since we are working with subsets of data. It is faster to train than decision trees because we are working only on a subset of features in this model, so we can easily work with hundreds of features. This gives accurate and precise results.

CODE FOR ALGORITHM:

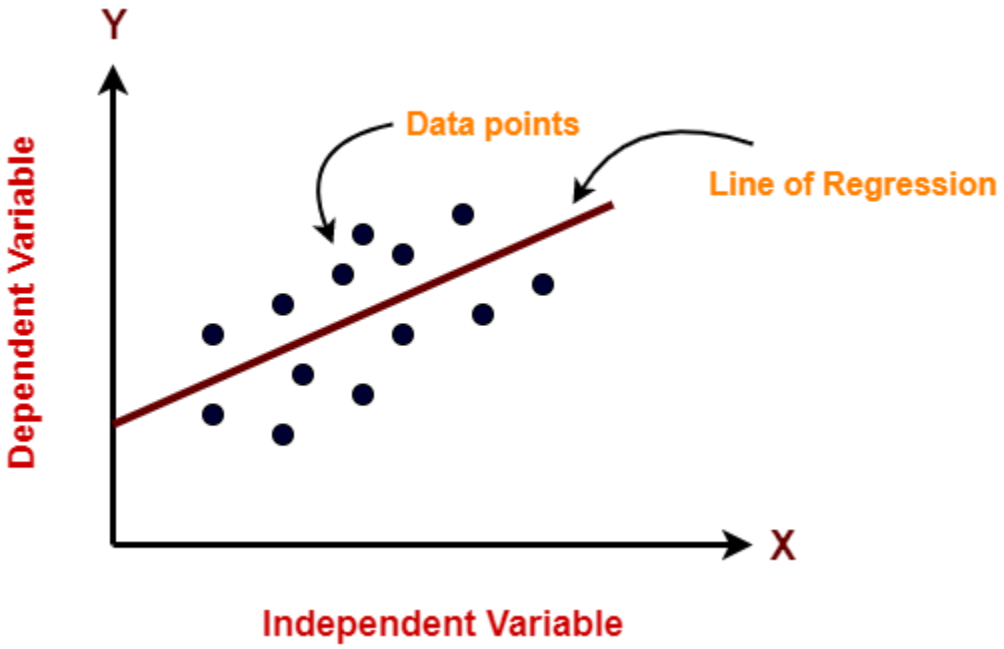
```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('')

#Extracting Independent and dependent Variable
x= data_set.iloc[].values
y= data_set.iloc[].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25)

#Fitting Random Forest Regressor to the training set
from sklearn.ensemble import RandomForestRegressor
regressor=RandomForestRegressor(n_estimators=10)
regressor.fit(x_train,y_train)

#Predicting the test set result
y_pred= regressor.predict(x_test)
y_pred
```

3.2 COMPARISON OF ALGORITHMS

LINEAR REGRESSION ALGORITHM

Linear regression is an algorithm that provides a linear relationship between an independent variable and a dependent variable to predict the outcome of future events. It is a statistical method used in data science and machine learning for predictive analysis.

The independent variable is also the predictor or explanatory variable that remains unchanged due to the change in other variables. However, the dependent variable changes with fluctuations in the independent variable. The regression model predicts the value of the dependent variable, which is the response or outcome variable being analyzed or studied.



The equation for linear regression is

$y_i=\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+...+\beta_p x_{ip}+\epsilon$

where,

for i=n observations

y_i=dependent variable

x_i=explanatory variables

β_0=y-intercept (constant term)

β_p=slope coefficients for each explanatory variable

ϵ=the model's error term

ADVANTAGES:

- Linear Regression performs well when the dataset is linearly separable. We can use it to find the nature of the relationship among the variables.
- Linear Regression is easier to implement, and interpret and very efficient to train.
- Linear Regression is prone to over-fitting but it can be easily avoided using some dimensionality reduction techniques, regularization techniques, and cross-validation.

DISADVANTAGES:

- The main limitation of Linear Regression is the assumption of linearity between the dependent variable and the independent variables. In the real world, the data is rarely linearly separable. It assumes that there is a straight-line relationship between the dependent and independent variables which is incorrect many times.
- Prone to noise and overfitting: If the number of observations is lesser than the number of features, Linear Regression should not be used, otherwise it may lead to overfitting because it starts considering noise in this scenario while building the model.
- Prone to outliers: Linear regression is very sensitive to outliers (anomalies). So, outliers should be analyzed and removed before applying Linear Regression to the dataset.
- Prone to multicollinearity: Before applying Linear regression, multicollinearity should be removed (using dimensionality reduction techniques) because it assumes that there is no relationship among independent variables.

CODE FOR ALGORITHM:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('')

#Extracting Independent and dependent Variable
x= data_set.iloc[].values
```

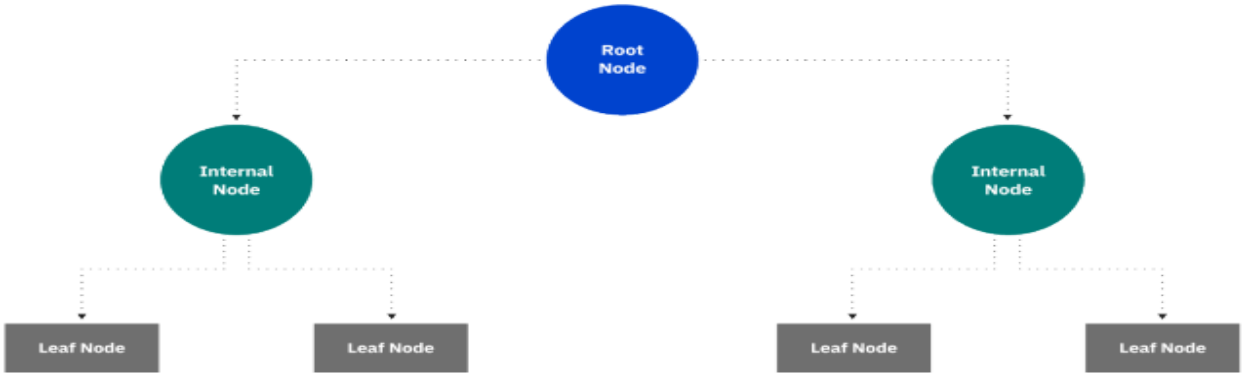
```
y= data_set.iloc[].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25)

#Fitting Linear Regression to the training set
from sklearn.ensemble import  LinearRegression
regressor= LinearRegression(n_estimators=10)
regressor.fit(x_train,y_train)

#Predicting the test set result
y_pred= regressor.predict(x_test)
y_pred
```

DECISION TREE REGRESSOR ALGORITHM

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems.The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.



ADVANTAGES:

- Compared to other algorithms decision trees requires less effort for data preparation during pre-processing.
- A decision tree does not require the normalization of data.
- A decision tree does not require the scaling of data as well.
- Missing values in the data also do not affect the process of building a decision tree to any considerable extent.
- A Decision tree model is very intuitive and easy to explain to technical teams as well as stakeholders.

DISADVANTAGES:

- A small change in the data can cause a large change in the structure of the decision tree causing instability.
- For a Decision tree sometimes calculation can go far more complex compared to other algorithms.
- The decision tree often involves higher time to train the model.
- Decision tree training is relatively expensive as the complexity and time have taken are more.
- The Decision Tree algorithm is inadequate for applying regression and predicting continuous values.

CODE FOR ALGORITHM:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('')

#Extracting Independent and dependent Variable
x= data_set.iloc[].values
y= data_set.iloc[].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25)

#Fitting Decision Tree Regressor to the training set
from sklearn.ensemble import DecisionTreeRegressor
regressor=DecisionTreeRegressor(n_estimators=10)
regressor.fit(x_train,y_train)

#Predicting the test set result
y_pred= regressor.predict(x_test)
y_pred
```

COMPARING THE MODEL:

The R-squared is often called the coefficient of decision. The proportion of variance is estimated from the independent variables in the dependent variable.
R-squared =Explained variance/Total variance
The more R-squared, the better the model output. , and indicates that the model deviates less from real values. An R-squared score of 1 indicates that it suits perfectly.
We used a linear regression algorithm and decision tree algorithm for comparing the model. These models are trained based on training data and tested on test data.
Using the above measures, we compared different models and the results are tabulated below:

Linear Regression	0.74
Decision Tree Regressor	0.71
Random Forest Regressor	0.83

3.3 PROGRAM CODE

3.3.1 Importing libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

3.3.2 Importing Datasets

```
# loading the data from csv file to a Pandas DataFrame
insurance_dataset = pd.read_csv('/content/insurance.csv')
```

3.3.3. Data Collection & Analysis

```
# first 5 rows of the dataframe
insurance_dataset.head()

# number of rows and columns
insurance_dataset.shape

# getting some informations about the dataset
insurance_dataset.info()

# checking for missing values
insurance_dataset.isnull().sum()

# statistical Measures of the dataset
insurance_dataset.describe()
```

```
# distribution of age value
sns.set()
plt.figure(figsize=(6,6))
sns.distplot(insurance_dataset['age'])
plt.title('Age Distribution')
plt.show()

# Gender column
plt.figure(figsize=(6,6))
sns.countplot(x='sex', data=insurance_dataset)
plt.title('Sex Distribution')
plt.show()
insurance_dataset['sex'].value_counts()

# bmi distribution
plt.figure(figsize=(6,6))
sns.distplot(insurance_dataset['bmi'])
plt.title('BMI Distribution')
plt.show()

# children column
plt.figure(figsize=(6,6))
sns.countplot(x='children', data=insurance_dataset)
plt.title('Children')
plt.show()
insurance_dataset['children'].value_counts()

# smoker column
plt.figure(figsize=(6,6))
sns.countplot(x='smoker', data=insurance_dataset)
plt.title('smoker')
plt.show()
insurance_dataset['smoker'].value_counts()

# region column
plt.figure(figsize=(6,6))
sns.countplot(x='region', data=insurance_dataset)
plt.title('region')
plt.show()
insurance_dataset['region'].value_counts()

# distribution of charges value
plt.figure(figsize=(6,6))
sns.distplot(insurance_dataset['charges'])
plt.title('Charges Distribution')
```

```
plt.show()
```

3.3.4 Data Pre-Processing

```
# encoding sex column
insurance_dataset.replace({'sex':{'male':0,'female':1}}, inplace=True)
 # encoding 'smoker' column
insurance_dataset.replace({'smoker':{'yes':0,'no':1}}, inplace=True)
# encoding 'region' column
insurance_dataset.replace({'region':{'southeast':0,'southwest':1,'northeast':2,'northwest':3}},
inplace=True)
```

3.3.5 Splitting the Features and Target

```
X = insurance_dataset.drop(columns='charges', axis=1)
Y = insurance_dataset['charges']
print(X)
print(Y)
```

3.3.6 Splitting the data into Training data & Testing Data

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
print(X.shape, X_train.shape, X_test.shape)
print(Y.shape, Y_train.shape, Y_test.shape)
```

3.3.7 Model Training

```
from sklearn.linear_model import LinearRegression
linear=LinearRegression()
linear.fit(X_train,Y_train)
predict=linear.predict(X_test)
from sklearn.metrics import r2_score
lr=r2_score(Y_test,predict)*100
print(lr)

from sklearn.tree import DecisionTreeRegressor
decision=DecisionTreeRegressor()
decision.fit(X_train,Y_train)
pred=decision.predict(X_test)
from sklearn.metrics import r2_score
lr2=r2_score(Y_test,pred)*100
print(lr2)

from sklearn.ensemble import RandomForestRegressor
random=RandomForestRegressor(n_estimators=50)
random.fit(X_train,Y_train)
```

```
predict2=random.predict(X_test)
from sklearn.metrics import r2_score
lr3=r2_score(Y_test,predict2)*100
print(lr3)

print('linear regression',lr)
print('decision tree',lr2)
print('random forest regressor',lr3)
```

3.3.8 Predicting the charge

```
def prediction():
  age=int(input("Enter the age: "))
  sex=int(input("Enter the sex  (0:male, 1:female): "))
  bmi=float(input("Enter the body mass index: "))
  children=int(input("Enter the number of children: "))
  smoker=int(input("Enter whether he/she is a smoker (yes:0, no:1): "))
  region=int(input("Enter the region (0:SE, 1:SW, 2:NE, 3:NW): "))

  input_data = (age,sex,bmi,children,smoker,region)
  # changing input_data to a numpy array
  input_data_as_numpy_array = np.asarray(input_data)
  # reshape the array
  input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)
  price= gradient.predict(input_data_reshaped)
  if (sex>1):
    print("The entered data is not valid")
  elif (smoker>1):
    print("The entered data is not valid")
  elif (region>3):
    print("The entered data is not valid")
  else:
    print('The insurance cost is USD ', price[0])

price=prediction()
Price
```

4 CHAPTER 3

4.1 RESULTS OF DATA ANALYSIS AND INTERPRETATION

```
# distribution of age value
sns.set()
plt.figure(figsize=(6,6))
sns.distplot(insurance_dataset['age'])
plt.title('Age Distribution')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distribut
  warnings.warn(msg, FutureWarning)
```

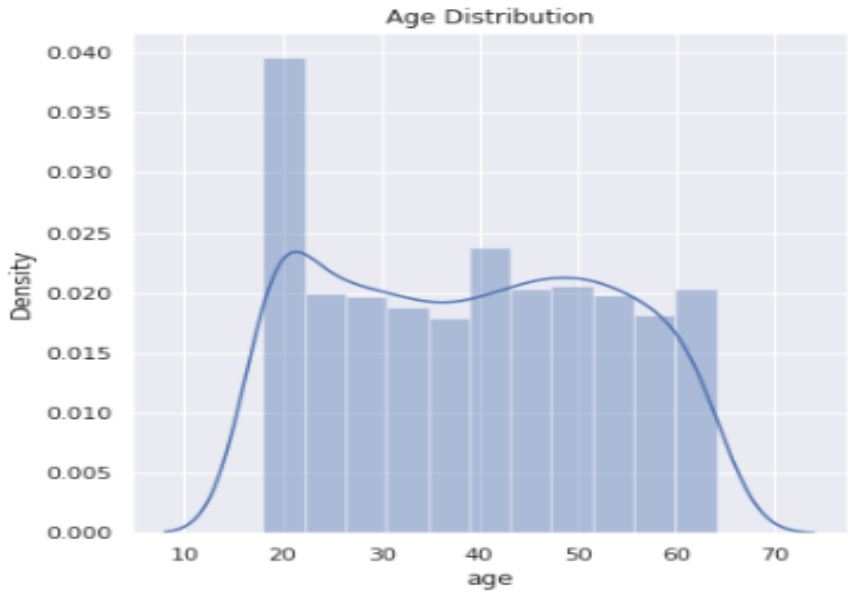


Fig 1.1 DISTRIBUTION PLOT OF AGE DISTRIBUTION

The distribution is from the age value from 10 to 70 and we have most number of values in the range of 20,21 and 22 .whereas maybe the values from 23 or 24 to about 70 t the distribution is almost equal in this age range. So it means more people in our dataset are at a particular age between 20,21 and 22.

```
# Gender column
plt.figure(figsize=(6,6))
sns.countplot(x='sex', data=insurance_dataset)
plt.title('Sex Distribution')
plt.show()
```

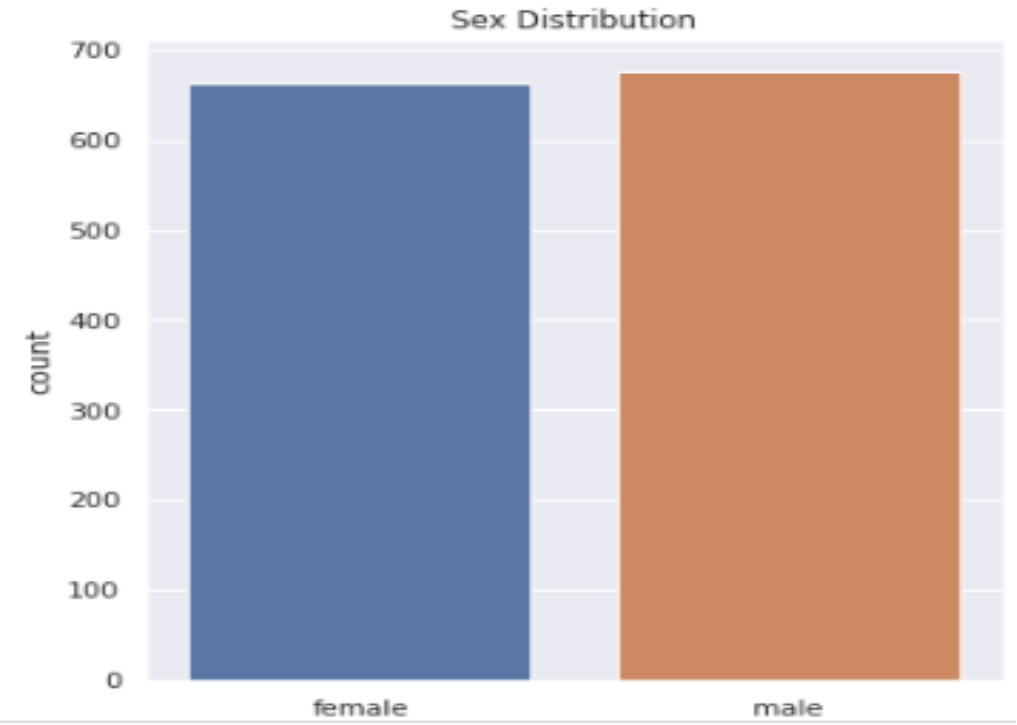


Fig 1.2 BAR GRAPH OF GENDER DISTRIBUTION

This graph represents the distribution of the number of males and females in the gender column. This graph shows that the number of males is almost equal to the number of females in the dataset.

```
# bmi distribution
plt.figure(figsize=(6,6))
sns.distplot(insurance_dataset['bmi'])
plt.title('BMI Distribution')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distribu
  warnings.warn(msg, FutureWarning)
```

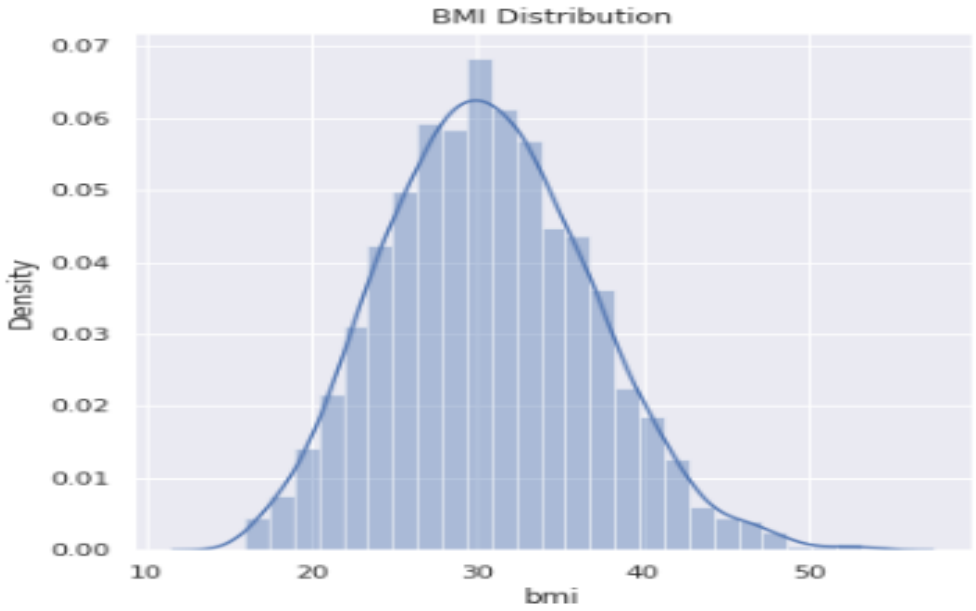


Fig 1.3 DISTRIBUTION PLOT OF BMI DISTRIBUTION

The more number of values are in this mid-range which is 30 and we have a similar kind of distribution on either side of the peak. So from this from 15 to 25, the BMI is increasing and more values around this 30 value. After bmi value 30 it kind of reducing.

```
# children column
plt.figure(figsize=(6,6))
sns.countplot(x='children', data=insurance_dataset)
plt.title('Children')
plt.show()
```

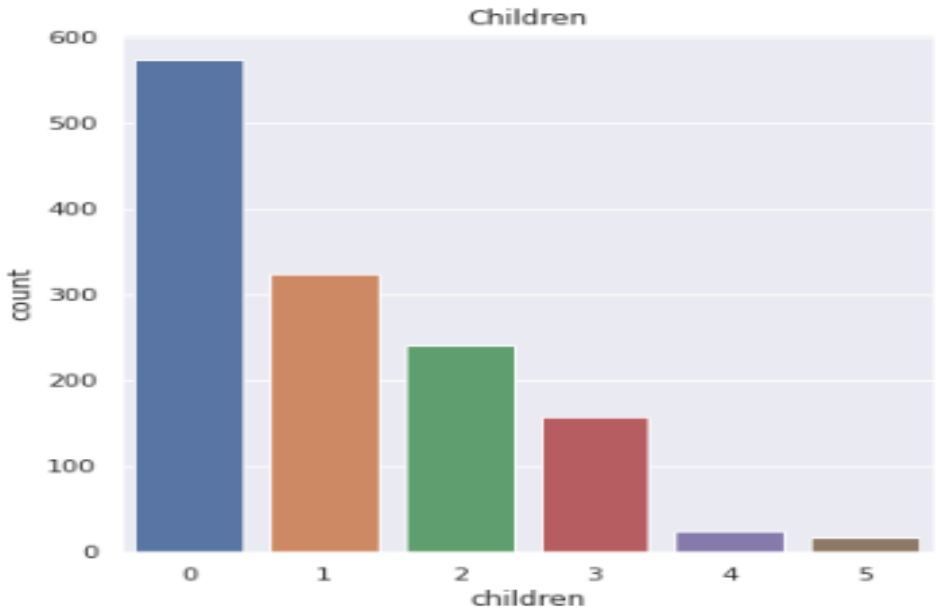


Fig 1.4 BAR GRAPH OF CHILDREN DISTRIBUTION.

It shows that more number of people who don't have any children and the count is around 570 and the number of people with one child is more compared to other values

```
# smoker column
plt.figure(figsize=(6,6))
sns.countplot(x='smoker', data=insurance_dataset)
plt.title('smoker')
plt.show()
```

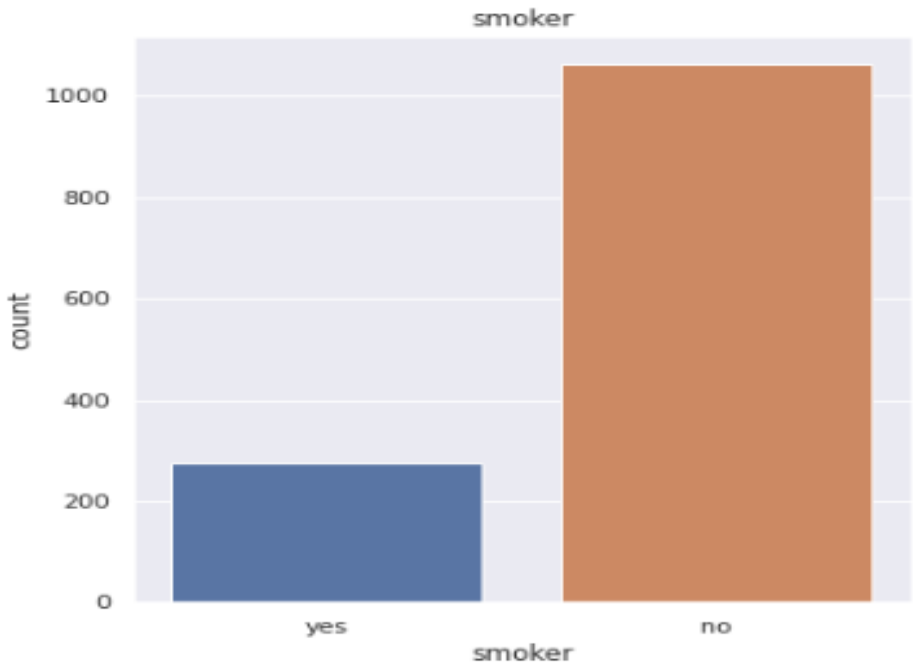


Fig 1.5 BAR GRAPH OF SMOKER DISTRIBUTION

It shows that the number of non-smokers is kind of more in this data set about more than a thousand and we have numbers of smokers of about between 200 to 250 value.

```
# region column
plt.figure(figsize=(6,6))
sns.countplot(x='region', data=insurance_dataset)
plt.title('region')
plt.show()
```

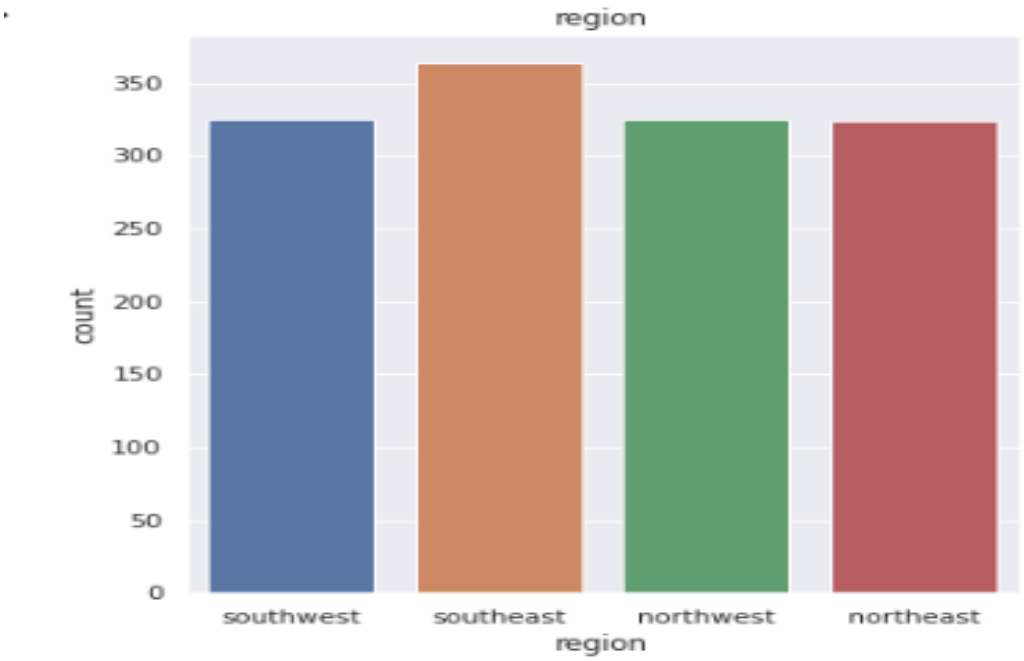


Fig 1.6 BAR GRAPH OF REGION DISTRIBUTION

The data is almost similar for all different types of regions. It is a kind of southeast region have more value compared to other regions.

```
# distribution of charges value
plt.figure(figsize=(6,6))
sns.distplot(insurance_dataset['charges'])
plt.title('Charges Distribution')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distrib
  warnings.warn(msg, FutureWarning)
```

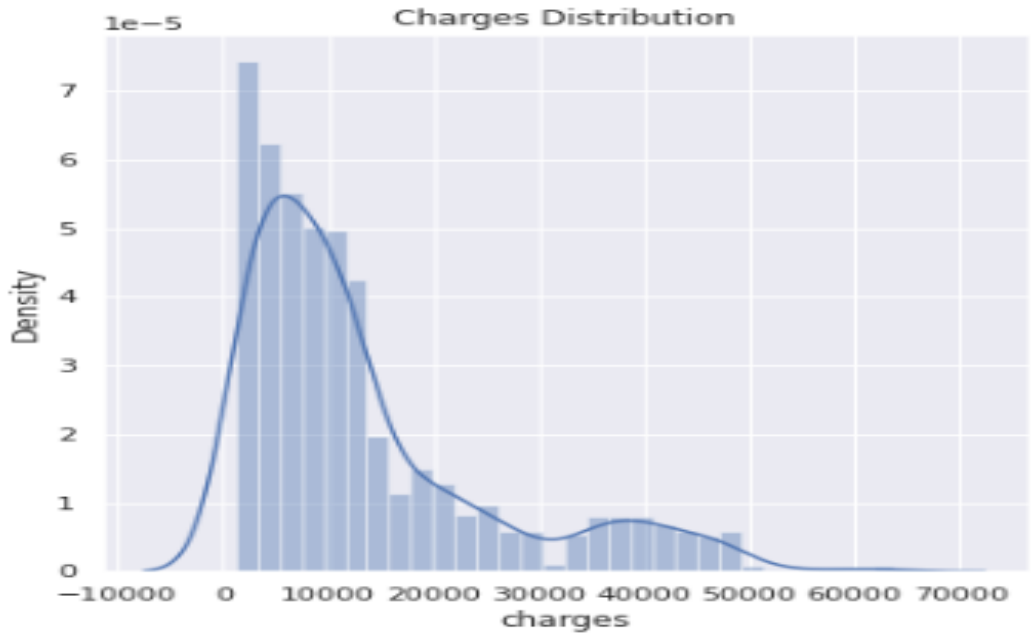


Fig 1.7 DISTRIBUTION PLOT OF CHARGE DISTRIBUTION

It shows that the more values are distributed in ten thousand dollars and there are very few values around thirty thousand and forty thousand dollars

ENCODING CATEGORICAL DATA INTO NUMERICAL DATA:

```
# encoding sex column
insurance_dataset.replace({'sex':{'male':0,'female':1}}, inplace=True)


 # encoding 'smoker' column
insurance_dataset.replace({'smoker':{'yes':0,'no':1}}, inplace=True)


# encoding 'region' column
insurance_dataset.replace({'region':{'southeast':0,'southwest':1,'northeast':2,'northwest':3}}, inplace=True)
```

Fig 2.1 Encoding Categorical Data Into Numerical Data

In the sex column, the values are male and female. We replace it with the values 0 and 1 respectively using replace() function. In the smoker column, the values are yes and no. We replace it with the values 0 and 1 respectively using replace() function. In the region column, the values are southeast, southwest, northeast, and northwest. We replace it with the values 0,1,2 and 3 respectively using replace() function

DEPENDENT AND INDEPENDENT VARIABLES

```
X = insurance_dataset.drop(columns='charges', axis=1)
Y = insurance_dataset['charges']
```

Fig 3.1 Dependent And Independent Variables

```
print(X)
```

	age	sex	bmi	children	smoker	region
0	19	1	27.900	0	0	1
1	18	0	33.770	1	1	0
2	28	0	33.000	3	1	0
3	33	0	22.705	0	1	3
4	32	0	28.880	0	1	3
...	...	...	...	...	...	...
1333	50	0	30.970	3	1	3
1334	18	1	31.920	0	1	2
1335	18	1	36.850	0	1	0
1336	21	1	25.800	0	1	1
1337	61	1	29.070	0	0	3

[1338 rows x 6 columns]

Fig 3.2

We define X as independent variables. Here, the independent variables are age, sex,bmi, children, smoker, and region.

.

```
print(Y)
```

```
0         16884.92400
1          1725.55230
2          4449.46200
3         21984.47061
4          3866.85520
              ...
1333      10600.54830
1334       2205.98080
1335       1629.83350
1336       2007.94500
1337      29141.36030
Name: charges, Length: 1338, dtype: float64
```

Fig 3.2

We define Y as the dependent variable. Here, the dependent variable is charges

SPLITTING THE DATASET

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

Fig 4.1

We split the data into training data and test data. X_train and Y_train are training data, and train size is 80%. X_test and Y_test are test data and test size is 20%.

```
print(X.shape, X_train.shape, X_test.shape)
(1338, 6) (1070, 6) (268, 6)
```

Fig 4.2

The size of X_train data is 1070 rows and 6 columns and X_test data are 268 rows and 6 columns

```
print(Y.shape, Y_train.shape, Y_test.shape)
(1338,) (1070,) (268,)
```

Fig 4.3

The size of Y_train data is 1070 rows and Y_test data is 268 rows.

ACCURACY OF ALGORITHMS

```
from sklearn.linear_model import LinearRegression
linear=LinearRegression()
linear.fit(X_train,Y_train)
predict=linear.predict(X_test)

from sklearn.metrics import r2_score
lr=r2_score(Y_test,predict)*100
print(lr)

74.47273869684076
```

Fig 5.1 Linear Regression

Linear regression performs the task to predict a dependent variable(target) based on the given independent variable. The accuracy of linear regression in this model is 74.47273869684076%

```
from sklearn.tree import DecisionTreeRegressor
decision=DecisionTreeRegressor()
decision.fit(X_train,Y_train)
pred=decision.predict(X_test)

from sklearn.metrics import r2_score
lr2=r2_score(Y_test,pred)*100
print(lr2)

71.17531494531374
```

Fig 5.2 Decision Tree Regressor

Decision tree regression observes the features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. The accuracy of the decision tree regressor in this model is 71.1753149453174%

```
from sklearn.ensemble import RandomForestRegressor
random=RandomForestRegressor(n_estimators=50)
random.fit(X_train,Y_train)
predict2=random.predict(X_test)

from sklearn.metrics import r2_score
lr3=r2_score(Y_test,predict2)*100
print(lr3)

83.09350538931196
```

Fig 5.3 Random Forest Regressor

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The accuracy of the random forest regressor in this model is 83.09350538931196%

```
print('linear regression',lr)
print('decision tree',lr2)
print('random forest regressor',lr3)

linear regression 74.47273869684076
decision tree 71.17531494531374
random forest regressor 83.09350538931196
```

Fig 5.4 Accuracy of the Model

From these model, the random forest regressor has the highest accuracy.

OUTPUT

```
def prediction():
  age=int(input("Enter the age: "))
  sex=int(input("Enter the sex  (0:male, 1:female): "))
  bmi=float(input("Enter the body mass index: "))
  children=int(input("Enter the number of children: "))
  smoker=int(input("Enter whelther he/she is a smoker (yes:0, no:1): "))
  region=int(input("Enter the region (0:SE, 1:SW, 2:NE, 3:NW): "))

  input_data = (age,sex,bmi,children,smoker,region)
  # changing input_data to a numpy array
  input_data_as_numpy_array = np.asarray(input_data)
  # reshape the array
  input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)
  price= random.predict(input_data_reshaped)
  if (sex>1):
    print("The entered data is not valid")
  elif (smoker>1):
    print("The entered data is not valid")
  elif (region>3):
    print("The entered data is not valid")
  else:
    print('The insurance cost is USD ', price[0])

price=prediction()
price
```

```
Enter the age: 20
Enter the sex  (0:male, 1:female): 1
Enter the body mass index: 22
Enter the number of children: 1
Enter whelther he/she is a smoker (yes:0, no:1): 0
Enter the region (0:SE, 1:SW, 2:NE, 3:NW): 2
The insurance cost is USD  15993.28758380001
```

Fig 6.1 Predicting The Charge

Define a function prediction().Enter the values of independent variables such as age, sex, bmi, children, smoker, and region from the user. Store this data into a tuple called input_data. Convert this tuple to a numpy array and store it into a variable called as_input_data_as_numpy_array. Then we reshape this array because we are predicting a data point and we store it into a variable called input_data_reshapedPredict the value using random. predict() and store it into a variable price.

Check the condition sex>1, then return invalid:

elif if smoker>1, then return invalid ;

elif if region>3, then return invalid:

otherwise, print the price

Call the function and assign price=prediction()

5 CONCLUSION

Machine learning (ML) is one aspect of computational intelligence that can solve different problems in a wide range of applications and systems when it comes to leveraging historical data. Predicting medical insurance costs is still a problem in the healthcare industry that needs to be investigated and improved. In this paper, by using a set of ML algorithms, a computational intelligence approach is applied to predict healthcare insurance costs. The medical insurance dataset was obtained from the KAGGLE website and was utilized for training and testing the Linear Regression, Decision Tree, and Random Forest Regressor ML algorithms. The regression of this dataset followed the steps of preprocessing, data splitting, regression, and evaluation. The resultant outcome revealed that Random Forest Regressor achieved a high accuracy of 83%.

In future work, we will use nature-inspired and metaheuristic algorithms to modify the parameters of machine learning and deep learning approaches on multiple medical health-related datasets.

6 REFERENCES

1. A Computational Intelligence Approach for Predicting Medical Insurance Cost, Ch. Anwar ul Hassan, Jawaid Iqbal, Saddam Hussain, Hussain AlSalman, Mogeeb A. A. Mosleh, and Syed Sajid Ullah
2. Predict Health Insurance Cost by using Machine Learning and DNN Regression Models, Mohamed Hanafy
3. Health Insurance Amount Prediction, Nidhi Bhardwaj, Rishabh Anand